A Portable 3D FFT Package for Distributed-Memory Parallel Architectures*

Hong Q. Ding† Robert D. Ferraro† Donald B. Gennery†

Abstract

A parallel algorithm for 3D FFTs is implemented as a series of local 1D FFTs combined with data transposes. This allows the use of vendor supplied (often fully optimized) sequential 1D FFTs. The FFTs are carried out in-place by using an in-place data transpose across the processors.

1 Introduction

Multi-dimensional FFTs are used frequently in engineering and scientific calculations, especially in image processing. Parallel implementations of FFT generally follow two approaches. One is the binary-exchange approach[1,2], where data exchanges take place in all pairs of processors with processor numbers differing by one bit. Another one is the transpose approach[2] for multi-dimensional FFTs, where a 3D FFT is carried out in 3 successive 1D local sequential FFTs with data transposes occurring in between. Inter-processor communication only take place in these data transpose. One advantage of this approach is that we can use the vendor-supplied 1D FFTs, which are often fully optimized. Furthermore, the data transpose part can use any general purpose parallel data transpose algorithm, thus easing the programming efforts significantly. Our 3D FFT package follows this approach. In particular, we follow a previous algorithm[3] closely, with the important difference that we devised an in-place data transpose algorithm, which reduces the computer memory to just one copy of the entire 3D data and a buffer for messages.

2 Data Distributions

Two choices of the 3D data distributions are provided. The first is a slab distribution (1D domain decomposition) in which a slab (a number of complete x-y planes) of the 3D data is distributed on each processor. 1D FFTs are carried out, for example, along the x and y dimensions. This is followed by a global data transpose to a data distribution in which each processor now contains complete x-z planes, so that FFTs along the z dimension can be carried out. Inverse FFTs follow the exactly reversed order.

In a rod distribution (2D decomposition), the x-y plane is evenly divided into all P processors. All data points along the z-dimension are in the same processor, thus forming a rod extending in z-direction. In this z-local distribution, 1D FFTs along z are carried out. A global data transpose then leads to a y-local distribution, and 1D FFTs along y are carried out. A second global data transpose then leads to x-local distribution and the 1D FFTs along x are carried out.

* Work funded by NASA HPCC ESS project.

† Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109.

3 In-place Global 3D Data Transposes

Our implementation for data transpose for data array$(N_x, N_y, N_z, \cdots)$ is a fairly general one, where N_x, N_y, N_z are not necessarily equal. For example, the same data transpose can be used in ADI solutions of PDE's.

The in-place algorithm first sets up a permutation table and then reshuffles data locally to the order expected when the global data transpose completes. In the communication phase, pairs of processors simply exchange these big data blocks. This completes the data transpose; the local data reshuffle already causes the data to be in correct order. This in-place transpose is slightly slower than the common two-copy data transpose[2] due to the small overhead of setting up the permutation table. Howeve, The memory saved in the our algorithm can make a crucial difference in many memory-bounded applications.

4 Timing and Scaling

The package has been implemented on the Intel Paragon, the Intel Touchstone Delta, and the Cray T3D, for slab distributions with all three coordinate orientations. Either real or complex input data in either double or single precision can be specified by the user. (Only 64-bit data is available on the T3D.) The rod distribution so far has been implemented only for some cases.

The figure uses some of the timing runs that have been made so far to show the scaling performance of the algorithm. It can be seen that the algorithm scales very well on the T3D, and fairly well on the Paragon and Delta. Accurate comparison of the absolute times on the different machines perhaps is not very meaningful, since the 1D FFTs are supplied by the vendor and may be coded differently on the different machines.

References

[1] G. C. Fox, et al.,*Solving Problems on Concurrent Processors*, Vol.I, Prentice Hall, Englewood Cliffs, NJ., 1988. pp.187-200.

[2] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing*, Benjamin/Cummings, Redwood City, CA, 1994. pp.377-405.

[3] E. P. Huang, P. C. Liewer, V. K. Decyk and R. D. Ferraro, *Concurrent Three-Dimensional Fast Fourier Transform Algorithms for Coarse-Grained Distributed Memory Parallel Computers*,

Caltech Report CRPC-93-10.